Universidad Nacional Autónoma de México

Facultad de Ingeniería

Laboratorio de Diseño Digital

Reporte de práctica 7

Decodificador BCD a 7 segmentos

Profesor:
M.I. Guevara Rodríguez Ma. del Socorro

Alumno(s):
Francisco Pablo Rodrigo

Grupo: 6

Calificación total _____

Previo _____

Desarrollo _____

Conclusiones _____

8 de abril de 2019

1. Objetivos

1.1. General

El alumno diseñará circuitos combinacionales (mediana escala de integración).

1.2. Particular

El alumno analizará, diseñará e implementará un decodificador para display de 7 segmentos.

2. Introducción

Un decodificador o descodificador es un circuito combinacional, cuya función es inversa a la del codificador, es decir, convierte un código binario de entrada (natural, BCD, etc.) de N bits de entrada y M líneas de salida (N puede ser cualquier entero y M es un entero menor o igual a 2^N), tales que cada línea de salida será activada para una sola de las combinaciones posibles de entrada.

Además, es un elemento digital que funciona a base de estados lógicos, con los cuales indica una salida determinada basándose en un dato de entrada característico, para el caso del decodificador de BCD a 7 segmentos,su función operacional se basa en la introducción a sus entradas de un número en código binario correspondiente a su equivalente en decimal para mostrar en los siete pines de salida establecidos para el integrado, una serie de estados lógicos que están diseñados para conectarse a un elemento alfanumérico en el que se visualizará el número introducido en las entradas del decodificador.

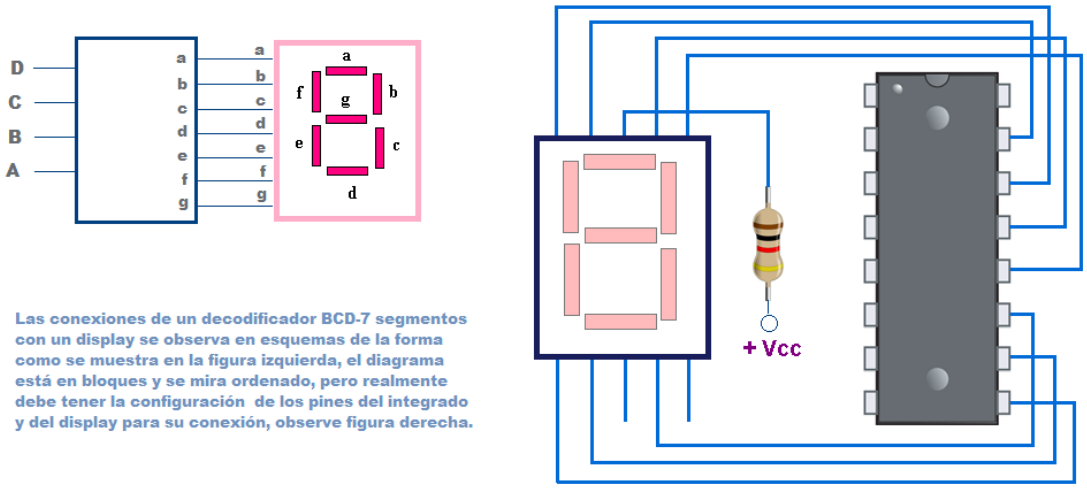

Figura 1: Simple bozquejo de como conectar el decodificador a un display de 7 segmentos

3. Previo

Recordemos que la tabla de verdad para para pasar de código binario a código de 7 segmentos es la siguiente

ENTRADA DEL DECODIFICADOR				SALIDAS DEL DECODIFICADOR						
D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	1	0	0

Figura 2: Código BCD

Recordemos que solo las primeras 10 combinaciones son válidas, del 1010 en adelante se les considera como *don't care*.

Por si las dudas también es importante recordar que el diagrama lógico de la conexión que se debería usando lógica negada.

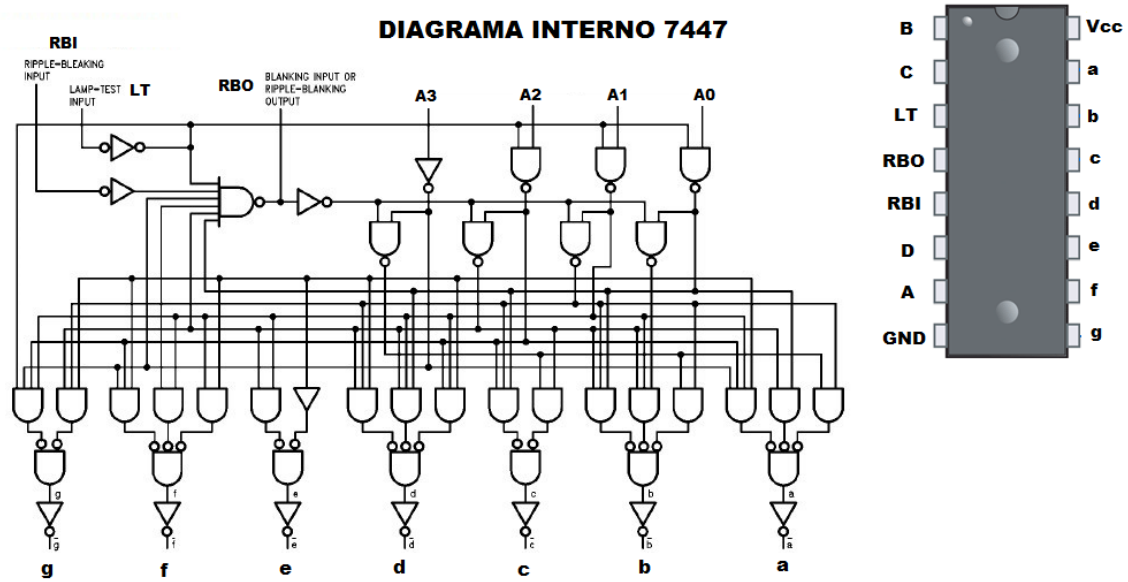

Figura 3: Diagrama lógico de BCD

3.1. Código en forma selectiva de VHDL

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity SEVEN_DISPLAY is
5    port (
6      estado    : in  std_logic_vector(6 downto 0);
7      display : out std_logic_vector(3 downto 0);
8      );
9  end SEVEN_DISPLAY;
10
11 architecture rtl of SEVEN_DISPLAY is
12 begin
13
14   WITH estado SELECT
15     display <= "0000001"    WHEN "0000",
16                "1001111"    WHEN "0001",
17                "0010010"    WHEN "0010",
18                "0000110"    WHEN "0011",
19                "1001100"    WHEN "0100",
20                "0100100"    WHEN "0101",
21                "1100000"    WHEN "0110",
22                "0001111"    WHEN "0111",
23                "0000000"    WHEN "1000",
24                "0001100"    WHEN "1001"
25                "1111111"    WHEN OTHERS;
26
27 end rtl;
```

4. Desarrollo

Lo primero que hicimos fue implementar lo que investigamos en el cuestionario Previo, de manera obtuvimos lo siguiente.

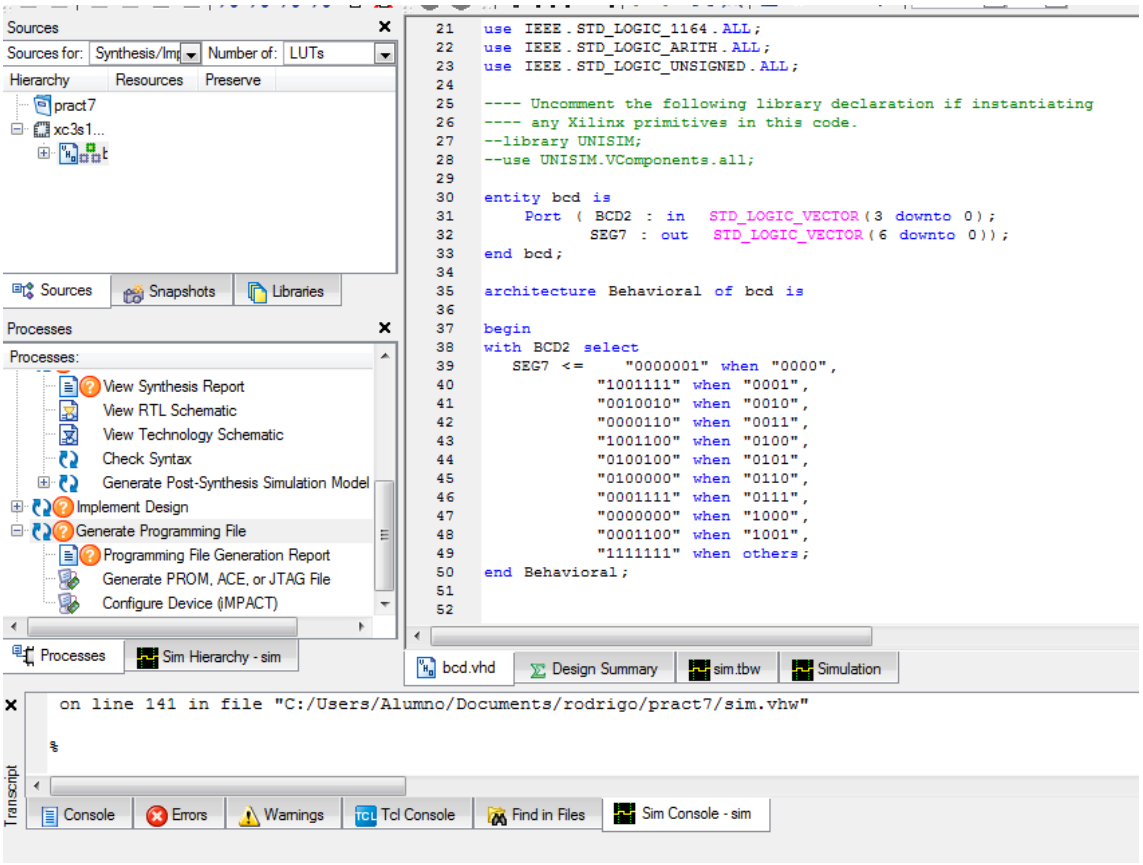

Figura 4: Implementación del circuito en VHDL

Una vez que checamos la sintaxis del código y de que todo salió bien, procedemos a simular. Esta vez la única diferencia es que por defecto tendremos la opción de *circuito combinacional*. La manera de simular los vectores es casi identica a como lo haciamos con simples variables, como se observa en la siguiente imagen.

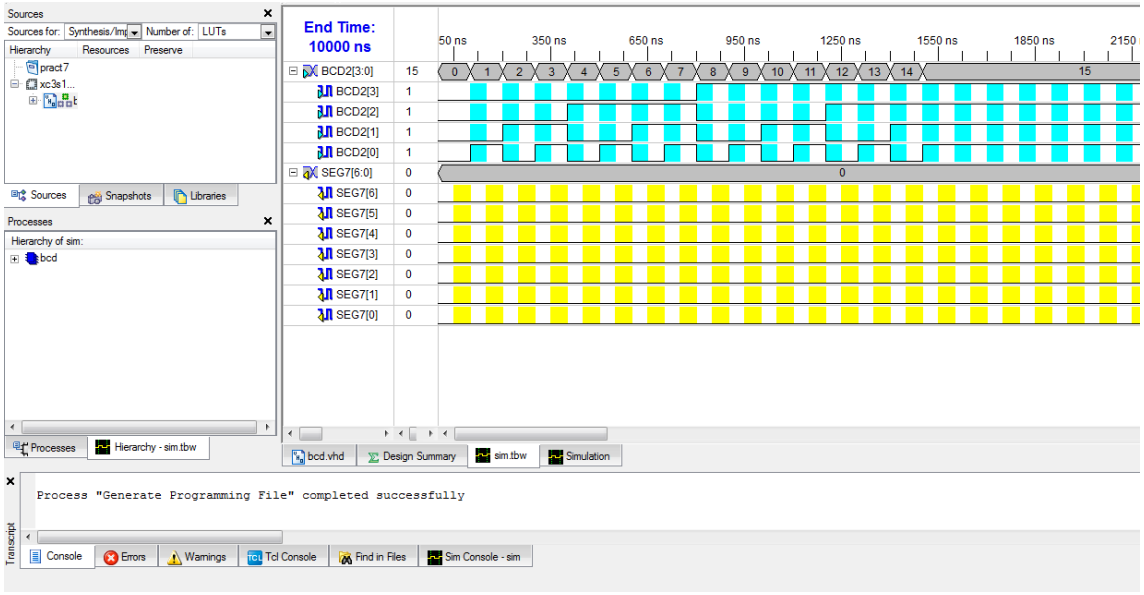

Figura 5: Configuración de la simulación

Después procedemos a generar nuestro archivo de simulación y obtenemos la siguiente respuesta. Observamos que al implemntar nuestro diseño con variables vectores obtenemos una salida más amena. Obtenemos directamente el número en decimal en lugar de binario.

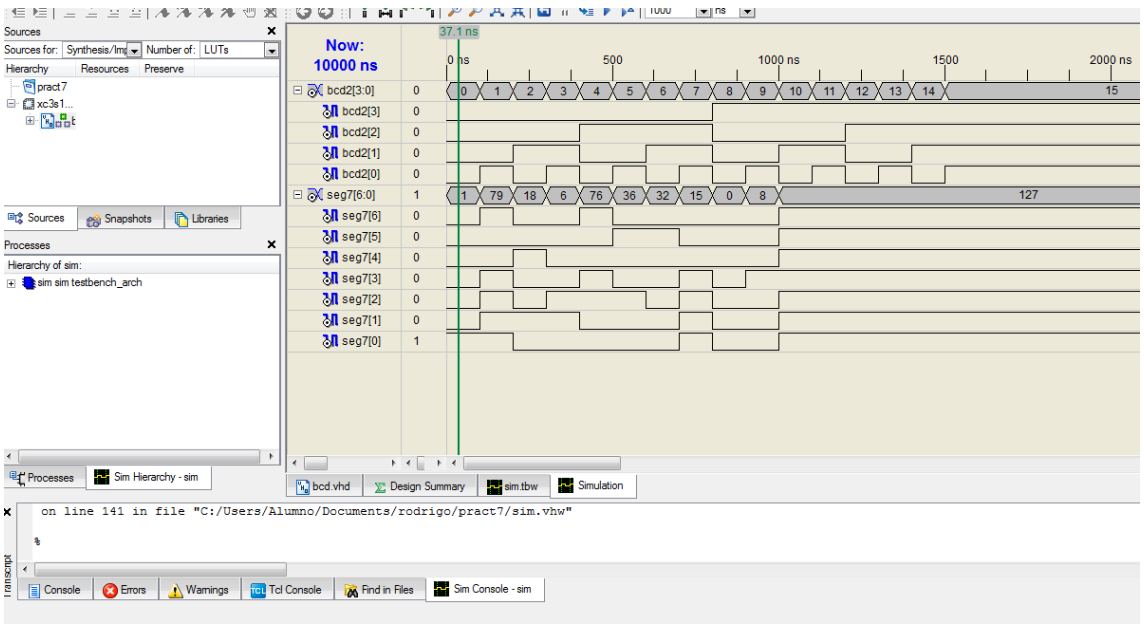

Figura 6: Simulación del circuito

Finalmente, cargamos nuestro circuito en la FPGA, para nuestro caso utilizamos un *Basys2*. Como recordatorio, el archivo que cargamos a la FPGA tiene terminación .bit

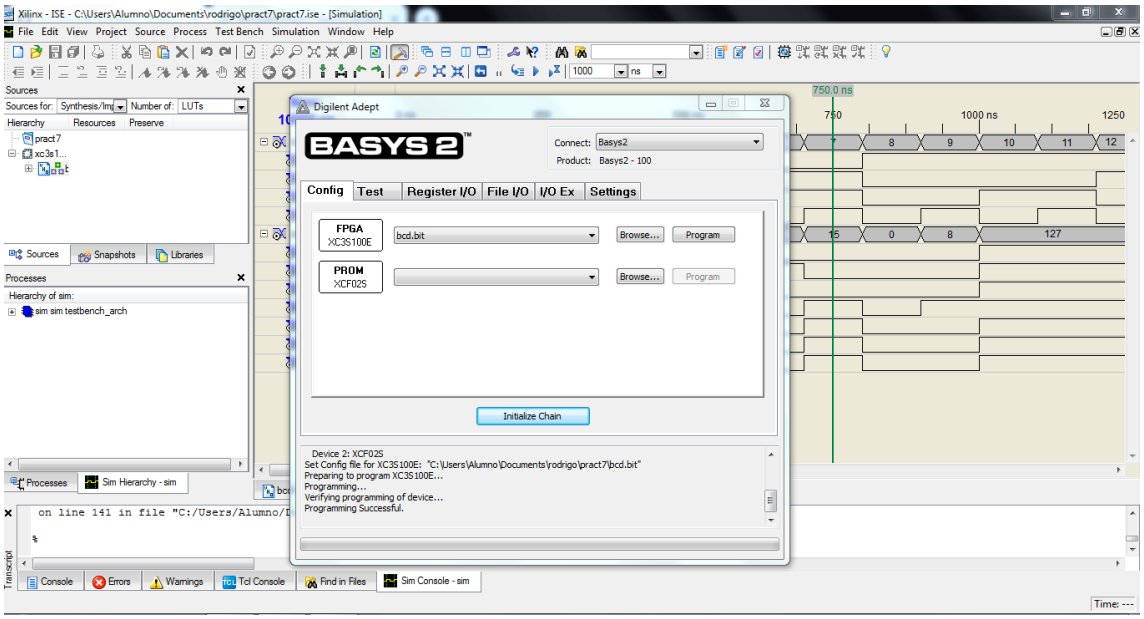

Figura 7: Con ADEPT cargamos el programa a la tarjeta

Por último probamos con algunas combinaciones para ver si el resultado es correcto.

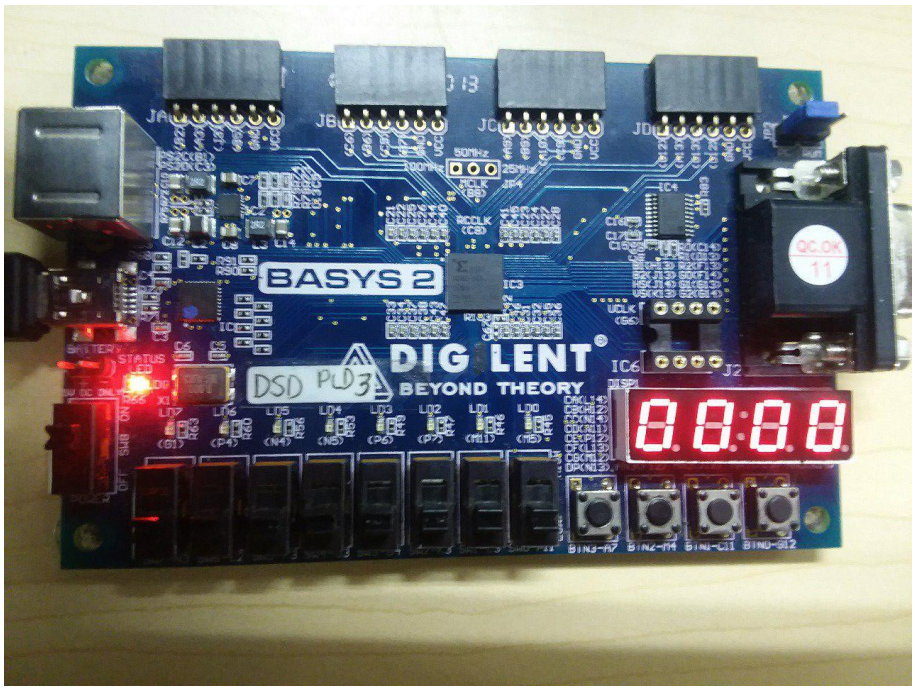

Figura 8: Circuito funcionando para la Configuración cero en binario

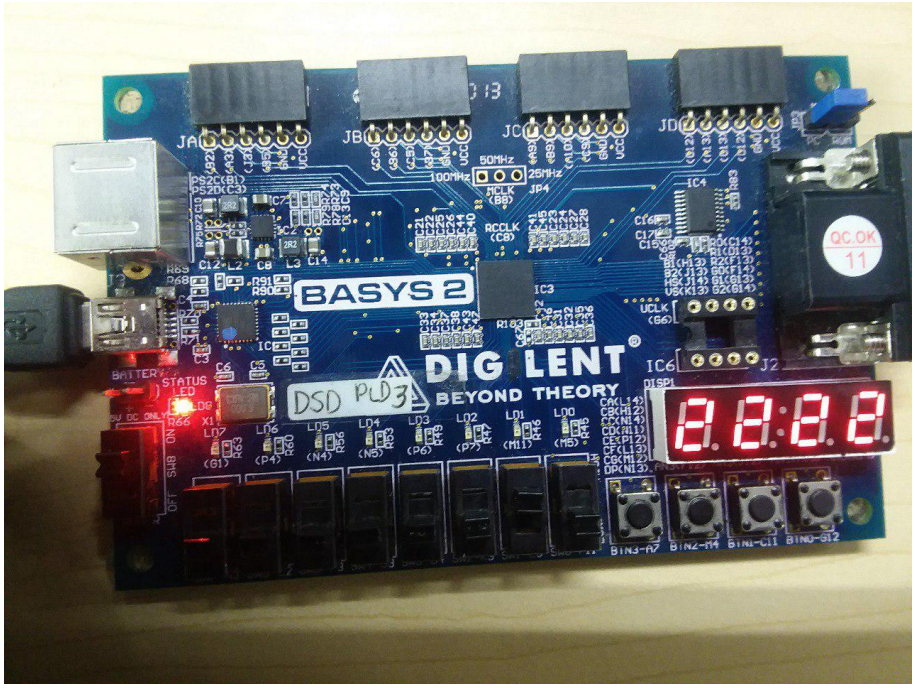

Figura 9: Circuito funcionando para la Configuración dos en binario

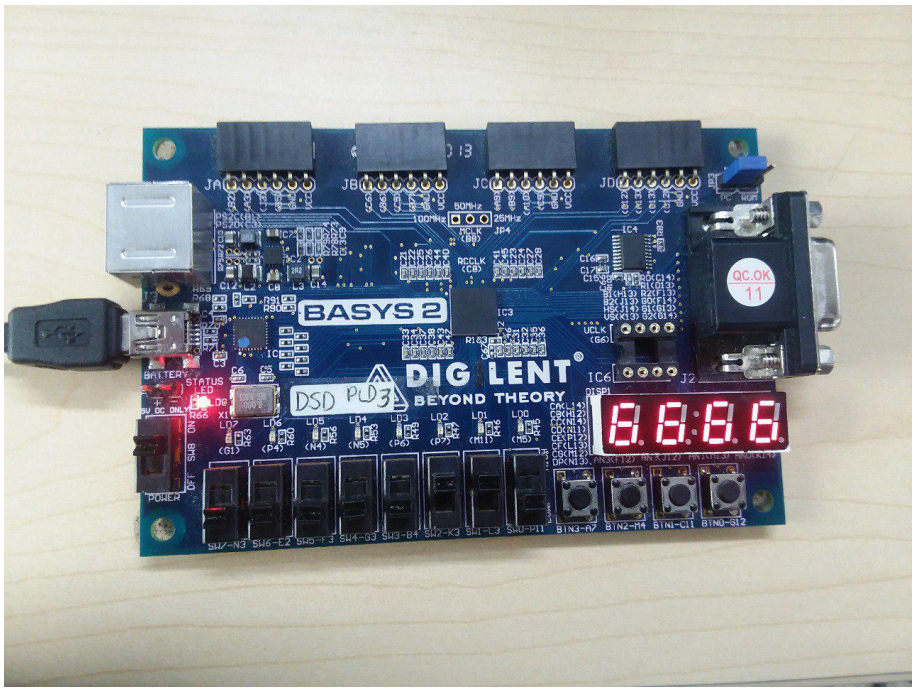

Figura 10: Circuito funcionando para la Configuración seis en binario

5. Conclusiones

El diseño de cualquier circuito combinacional se hace más sencillo ahora que tenemos dominadas otras herramientas de lenguaje VHDL, los cuales son variables en forma de vectores y sentencias de control de flujo como lo es el "with/select". Dichas herramientas nos ahorran la necesidad de generar decenas de compuertas OR y AND que son las que nos permitirían comunicar nuestro circuito con el display de 7 segmentos.

Observamos que esta práctica estuvo sencilla, sin embargo, a partir de ésta podemos crear cosas con mayor dificultad en cuanto a lógica se refiere.

También recordamos que es necesario saber si nuestro circuito presenta lógica negada o no, dicha información la podemos encontrar en la documenctación de la FPGA.